Rendering Procedural Textures for Visualization of Thematic Data in 3D Geovirtual Environments

Matthias Trapp, Frank Schlegel, Sebastian Pasewaldt and Jürgen Döllner
Hasso Plattner Institute, University of Potsdam, Germany
{*matthias.trapp, sebastian.pasewaldt, juergen.doellner*}*@hpi.de*

Keywords: 3D Geovirtual Environments, Procedural Texturing, Thematic Data, Real-Time Rendering

Abstract: 3D geovirtual environments, such as virtual 3D city and landscape models, can be used as scenery for visualizing thematic data. which can be communicated using suitable color mappings or hatch patterns. For rendering purposes, these hatches patterns can be represented as image-based or procedural textures. The resulting quality of image-based textures, and thus the effective communication of the respective thematic data, is subject to resolution and filtering artifacts. In contrast to thereto, procedural textures are not limited with respect to resolution and can be filtered adaptively to achieve high visual quality. However, challenges in parametrization and design often hinders their application. To counterbalance these drawbacks, this paper presents an interactive rendering technique that facilitates the application and design of procedural hatch patterns for the mapping of thematic data to 3D geovirtual environments.

1 INTRODUCTION

In 1983, Bertin described the idea of disassembling visual information into seven basic components he denoted as *visual variables*, such as position, size, and color (Bertin, 1983). These visual variables can be combined to form specific patterns. A *pattern* can be defined as a repetitive surface appearance that is characterized by one or more visual variables.

A *hatch pattern* is a special kind of pattern that is composed of regular strokes. The significant distinguishing features of hatch patterns are *size*, *orientation*, and *texture*. Most of these hatch pattern (also known as hatches or hachures) have evolved historically due to restrictions of the paper medium. Nevertheless, they can provide a sufficient distinction of nominal data and enable a non-ambiguous mapping. For example, thematic maps make use of hatch patterns that represent certain feature types. In specific application domains, these pattern are often standardized (e.g., DIN ISO 128-50).

Despite its application in 2D cartography, hatching is sparsely used in 3D geovirtual environments (3D GeoVEs), which can serve as scenery for communication and visualization of geo-referenced data, because of distortion effects caused by perspective projections. Since the application of perspective projections implicitly define other visual variables (e.g., size and order), hatch patterns can be considered challenging in 3D GeoVE specifically. Further, they are useful in cases where, for example, color-only mappings are not sufficient enough or not possible (e.g., monochromatic displays). Many 3D GeoVEs, such as landscapes and their generalized variants (Glander and Döllner, 2007) feature a number of planar surfaces that are suitable to display hatch patterns. Therefore, they are potentially suited for using hatch textures for data visualizing data as shown in Figure 1. It displays a virtual 3D model of the Grand Canyon with hatch textures that provide information about the land usage.

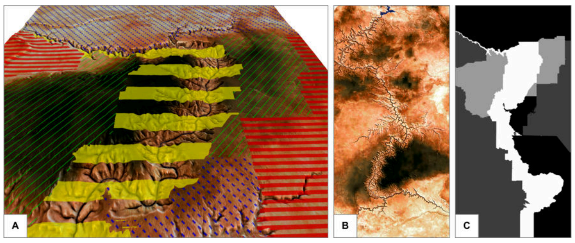


Figure 1: Examples for procedural hatch patterns applied the a virtual 3D landscape model of the Grand Canyon (A). Different colored hatch patterns are synthesized for different parts of the model (C) and blended with an aerial image of the regions (B).

There are basically two approaches to represent hatches for rendering in 3D GeoVE: *image-based textures* or *procedural-based textures.* While image-based or textures are easy to create, manipulate, and render, they also exhibit the major disadvantage of having fixed spatial resolution, which can yield sampling artifacts. To counterbalance this, three approaches can be applied: first, one can use *image-based distance maps* (Green, 2007) to improve sampling. This approach requires a single texture per hatch and impacts memory consumptions for a high number of different patterns. Second, one can convert an image-based hatch texture to a *vector texture* representation encoded in raster-buffers. However, its preprocessing and implementation is complex. Third, *procedural textures* can be applied. In contrast to image-based textures, procedural-based textures are computed at runtime and not restricted in terms of resolution and sampling. These properties qualifies them for implementing hatch patterns in interactive 3D visualizations. Nevertheless, procedural texturing suffers from a trade-off between its visual complexity and complexity required for its description, i.e. the more complex a procedural should appear the longer is its code to describe. In terms of performance, this results in higher runtime-complexity, thus slower rendering. However, complex hatch patterns are hard to describe procedurally.

This paper describes a method for composing and rendering of hatch patterns in 3D GeoVE using procedural texturing. It introduces a layering concept for creating complex hatch patterns by combining layers of simpler ones. This also reduces code complexity and facilitates reuse. However, using hatch patterns in 3D GeoVE become a challenging task because modification of position and viewing angle of the virtual camera affect the on-screen appearance of patterns. This may result in distracting, unpleasant effects, e.g., Moiré Patterns (Amidror, 2009). With respect to this, the paper describes different techniques for counterbalancing such effects.

2 RELATED WORK

This section focus on recent research on and application of procedural textures in 3D GeoVE. Rost defines procedural texturing as "the process of computing a texture primarily by *synthesizing* rather than by relying heavily on precomputed values" (Rost, 2006). In contrast to image textures, procedural textures are computed at runtime using vertex and/or fragment coordinates.

Hatch pattern are of manifold applications in the domain of geovisualization and information visualization in general. In cartography for example, hatches are used for representing 3D topography on a 2D map by displaying quantitative measures of the topography's slope and aspect (Kennelly and Kimerling, 2000). Here, lines are drawn in the direction of the steepest topographic gradient. This creates tonal variations throughout the map, which are a form of analytical hill-shading, creating a 3D impression of the topography. Also geological illustrations in text books make use of hatches to illustrate seismic data. In (Patel et al., 2007; Patel et al., 2008) an approach is presented for rendering such illustrations. There are various techniques that can be used and combined to generate procedural textures. This paper's concept is based on propagating a 2D pattern to a 3D space, and thus generates a so called *solid texture* (Peachey, 1985). Prominent representatives of solid textures are wood, granite, or marble textures, that often use noise to create a natural look (Perlin, 1985; Lewis, 1989). For mapping a texture to a 3D object, the surface of that object must be parameterized with 2D texture coordinates. During the mapping process the color of a fragment is determined by mapping the fragment to a texel in the image texture using these coordinates. In contrast, solid textures use 3D (world) coordinates of a fragment as input for their color computation.

3 PROCEDURAL PATTERNS

Based on preliminaries and requirements, this section introduces a basic concept for hatching 3D objects in 3D GeoVEs.

3.1 Preliminaries

Assumptions & Requirements. To apply hatch patterns to features of a 3D GeoVE, it is necessary to enrich its geometry with additional per-vertex attributes. In a preprocessing step, an *unique object identifier ID* is computed, which enables the identification of a polygon in the programmable rendering pipeline during runtime. For mappings independent of geometric representation of features, e.g., per-pixel mapping of a virtual 3D landscape model, an *image-based id-texture* is used (Fig. 1.B). In addition, an axis-aligned bounding box (AABB) for each feature geometry is computed and stored as a per-vertex attribute. The AABB enables the computation of texture coordinates during rendering (Sec. 3.3).

Standard texture mapping (Akenine-Möller et al., 2008) that relies on per-vertex texture coordinates is not always suited for creating consistent hatch patterns. The results depend on a consistent texture parametrization of the object's surfaces. Such parametrization must be provided in advance and can be hard to compute. Texture coordinates that are not evenly spaced or discontinuous on object edges yield inconsistencies in the rendered pattern. Further, a hatch pattern should appear identically on each object it is applied to preserve a distinctive mapping between pattern and data, i.e., the pattern is not allowed to vary in scale or orientation between different objects. The presented approach uses a variant of projective texturing to address these two requirements. Furthermore, the patterns should be applicable on objects in interactive applications dynamically. Hence, the implementation must support real-time rendering as well as a flexible mapping between feature identifiers and hatch patterns. To achieve the latter, the approach enables the binding of a hatch pattern P to a number of specific identifiers (ID). This mapping is resolved during texturing using the programmable graphics pipeline (Segal et al., 2010).

Terminology. A *procedural texture* or *hatch pattern* $P \in \mathcal{P}$ of a feature or object instance with the unique identifier id is defined as a polymorphic list of *pattern layer* instances L_i^{type}:

$$P_{ID} = L_0^{type}, \ldots, L_n^{type} \quad L_i^{type} \in \mathcal{L} \quad L_i^{type} = \{p_{type}\}$$

The definition of layer instances can be shared between different hatch pattern. The set of all hatch definitions is denoted as $\mathcal{P}$ and the set of all layer instances as $\mathcal{L}$. The respective layer type can be one of the following: $type \in \{hatch, glyph, noise\}$. A procedural texture layer is defined using a set of type specific parameters p_{type}, which are discussed in the next section. To synthesize complex patterns, different instances of layer types can be combined to a single pattern. Therefore each layer is attributed by a specific Boolean combination functions (e.g., OR, AND, XOR), which enable arbitrary layer combinations.

3.2 Components of Procedural Patterns

We identified three main components, denoted as *layer types*, as the major building blocks of complex hatch patterns: *linear hatch layer*, *glyph layer*, and *noise layer*. Their respective parametrization are presented in the remainder of this section. Each layer type shares a common parameter set that comprises (1) two orthogonal vectors $\vec{u}, \vec{v} \in \mathbb{R}^3$ which define orientation of a *reference plane* for the 2D hatch pattern, (2) a combination operator (logical OR, XOR, AND), and (3) a color.

Linear Hatch Layer. A linear hatch layer denotes variants of linear hatch features such as solid or stippled lines. It is one of the most used primitive for representing hatches. Its parametrization comprises the following aspects $L_{hatch} = (s, w, p, t)$: a hatch scale factor s defines how many lines should occur within an interval, the hatch width w defines the line width in relation to the space between, a stipple pattern p represents a bit mask describing the pattern of the stipples, and a stipple scale factor t defines how often the stipple pattern should occur.

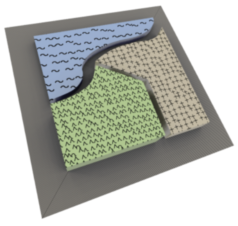


Figure 2: Examples of different glyphs layers applied to feature types.

Glyph Layer. A glyph layer L_{glyph} supports the creation of complex patterns that can be hardly represented by linear hatch layers, e.g., rounded shapes or symbols. Glyph layers can be represented directly using image-based textures or by distance-fields (Green, 2007) organized by texture atlases (Wloka, 2003). In contrast to image-based textures, distance fields provide significant visual improvements due to the lack of aliasing artifacts during up-sampling and reduces texture memory consumption. Glyph layers organized in textures atlases can be efficiently rendered using *texture bombing* or *glyph bombing* rendering techniques (Glanville, 2004).

Noise Layer. To add irregularity to a hatch pattern, a noise layer L_{noise} can be applied. Conceptually similar to a distance map, its represents gray-scale values $g \in [0, 1]$ that can be thresholded using a parameter $\varepsilon \in [0, 1]$ to convert it into a binary representation. Noise layer can be represented using (hardware-accelerated) noise functions (Lagae et al., 2010) or tileable noise textures (Perlin, 2002; Lewis, 1989). The design space comprises varying noise frequencies, a threshold, and a scale factor s for the generated texture coordinates. For in 3D virtual environments, anisotropic noise (Goldberg et al., 2008) can be used to minimize perspective artefacts (Sec. 3.4).

3.3 Texture-Coordinate Generation

To support changes in the mapping of object geometry to hatch patterns, the coordinates for texturing and evaluation are computed procedurally based on the AABB of each object. Here, a respective vertex position V_i is first normalized according to its axis-aligned bounding box $AABB_{id} = (LRF, URB)$, which is defined using the coordinates of its lower left front (LLF) and upper right back (URB) vectors:

$$V_i = \frac{V_i - LLF}{2 \cdot LLF + URB}$$

The resulting normalized coordinates are interpolated during rasterization and yield a texture coordinate for each fragment, which is then used to compute the individual hatches and their combination.

3.4 Counterbalance 3D Projections

A major issue of applying hatching to interactive 3D GeoVE is to ensure the perception of patterns – regardless of a 3D perspective projection transformation. For example, by zooming in and out, the line width and in-between line distances vary with an increasing distance to the virtual camera. This is due to the single continuous scale encountered in 3D GeoVE, instead of discrete scales in 2D visualizations (Jobst and Döllner, 2008). Further, by rotating the virtual camera, the hatch orientation can change. When tilting the virtual camera, the pattern will be distorted due to perspective compression. An improper aspect ratio can also distort the slope of a line so that the strokes look curved instead of straight.

All of the above may lead to ambiguity and an inaccurate mapping of patterns to data. On the one hand, it is naturally caused by perspective compression, on the other hand it hinders the correct and non-ambiguous communication of the information encoded in the patterns, i.e., distinguishing and comparing patterns at different scene depth becomes hard, especially for almost similar patterns. Further, if the in-between line distances of a hatch pattern becomes too small, it interferes with the screen raster. That causes a vibrant effect known as Moiré Pattern (Amidror, 2009) as shown in Figure 3. This effect is unpleasant, yields temporal incoherency, and makes it hard to identify the original pattern. The remainder of the section describes approaches for minimizing, counterbalancing, or avoiding these effects.

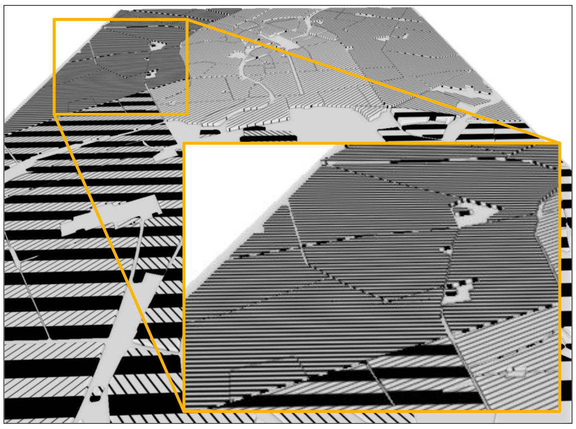


Figure 3: Decreasing distances between between linear hatches can result in Moiré patterns.

Fading Distant Hatch-Patterns. One approach for counterbalancing Moiré effects is to omit the rendering of hatch patterns in distant regions (Fig. 4(a). Here, the distance to the virtual camera can be thresholded (*fading distance*), and hatch patterns are faded by smoothing the hatch density. This approach is similar to the rendering of fog (Akenine-Möller et al., 2008). However, this approach has a number of limitations which prevents its application. Since the Moiré effect appears stronger for patterns with higher hatch frequency and larger hatch width, the effective fading distance depends on the respective hatch configuration used. This causes patterns of different frequency to have different fading distances, which create an inconsistent look-and-feel in the final visualization. Using a suitable threshold, this approach avoids the Moiré effect, but it will also provide more ambitiousness to the hatch mapping. Further, it is challenging to find an adequate threshold that avoids the creation of an Moiré effects but does not fade the pattern too early.

Depth-dependent Hatching. Another approach against the Moiré Effect is to avoid thick lines by scaling the hatch distance depending on the particular depth of the fragment. This can be achieved by computing the normalized depth value of the fragment and scale the pattern respectively. A result of that method is depicted in Figure 4(b). The pattern in the foreground remains the same as in Figure 4(a), but in the background the distance between the hatches is increased. By this means, the Moiré effect only occurs for low viewing angles. Another advantage of this technique is shown in Figure 5. While zooming, the distance between the lines in screen space remains the same, i.e, independent of the zoom level. It facilitates pattern recognition on every zoom level, which is otherwise not possible because the hatches become too small to be recognizable on a high zoom level. However, this method also has a disadvantage: The pattern is moving with the virtual camera, i.e., when the camera moves toward a fragment, the relative depth value of that fragment changes. Hence that fragment is either hatched or not, depending on the camera position.

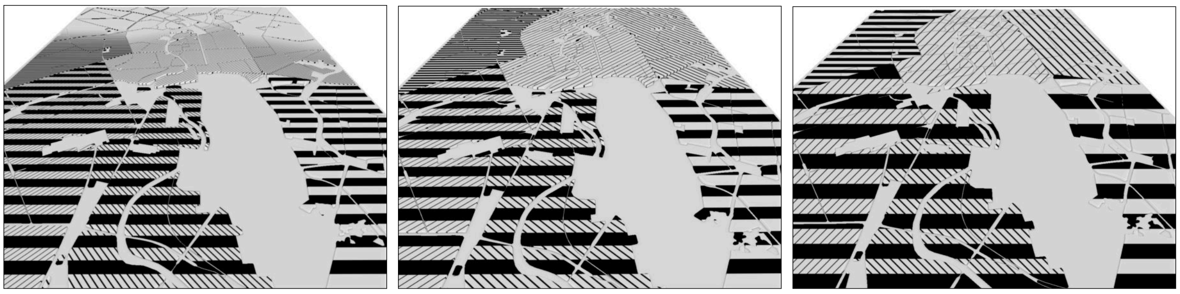

(a) Distance-based fading. (b) Depth-dependent hatching. (c) Screen-space hatches.

Figure 4: Three different approaches for compensating perspective distortion: distance-based fading (a), depth-dependent hatching (b) and based on screen-space coordinates (c).

Screen as Frame-of-Reference. An other approach represents the utilization of alternative texture coordinates: instead of generating 3D textures coordinates of the world space prior to geometry rasterization, the actual screen-space fragment coordinates can be used to generate the hatch pattern. A visualization using this approach is shown in Figure 4(c). The patterns on the surfaces always look identical–independent of zoom level, view angle or distance of the fragment. This technique provides sufficient perception and comparability of patterns and enables an unambiguous mapping of patterns to data. However, it also introduces the following texturing artifact: the hatches do not move consistently with the objects they are applied on during camera position changes. This causes the same effect as using the depth-dependent hatching. Further, a lack of perspective compression makes it hard to create an impression of depth in the scene. Furthermore, an additional shading model is required to visualize the edges and surface characteristics of 3D objects, because the pattern does not adapt to the objects' surfaces.

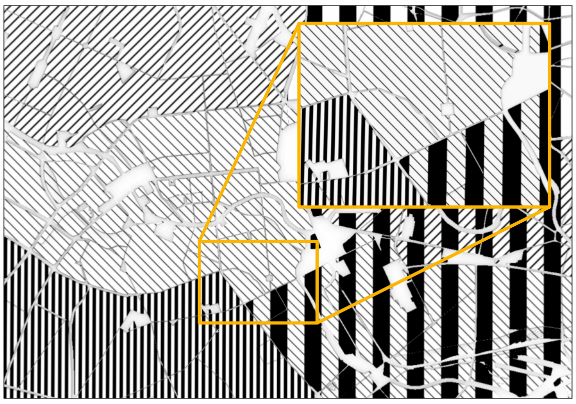


Figure 5: By scaling the hatch distance depending on the depth the pattern stays almost consistent during zooming.

4 RESULTS & DISCUSSION

This section presents application examples and evaluates their rendering performance.

Application Examples. The presented concept and rendering technique can be used in various applications Despite texturing of virtual 3D landscapes (Fig. 1), it is suitable for cell-based generalization variants of virtual 3D city models (Glander and Döllner, 2007). The abstraction of complex city geometry by creating generalized shapes yield a number of flat surfaces that are qualified for applying hatches to visualize data (Fig. 6). Another application of procedural hatches are 3D architectural models. Here, hatch pattern can be used to display building material according to a specific standard to match 2D construction plans. Figure 7(a) shows an explosion view of virtual 3D model of a reservoir dam with different linear hatch pattern to visualize different materials. It demonstrate the ability of the presented rendering technique to synthesis of solid textures (Peachey, 1985), i.e., hatch pattern that spread out consistently over a 3D object's surface. Figure 7(a) shows a geological profile with colored linear hatch pattern for the visualization of soil types.

Performance Evaluation. We evaluated a prototypical implementation based on OpenGL and the OpenGL Shading Language (GLSL) (Kessenich et al., 2010) using test data sets of different geometric complexity (Table 1), i.e., a varying number of scene objects and different numbers of hatch layers. The rendering and compositing of hatch layers is performed using a single fragment shader program.

The performance tests are conducted using an Intel Core i7 620M processor with 2,66 GHz clock rate and 8 GB of DDR3-1066 RAM using a NVIDIA GT 330M graphics card with 512 MB of video memory. The test application runs in windowed mode at two

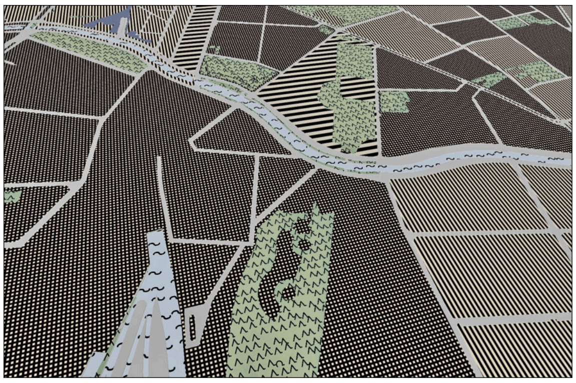
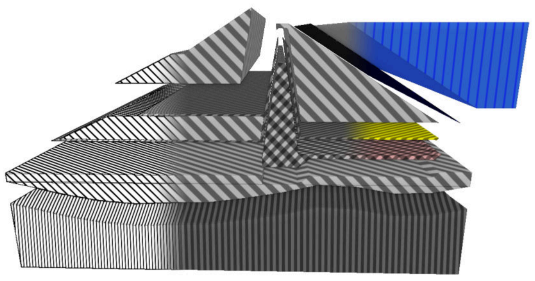


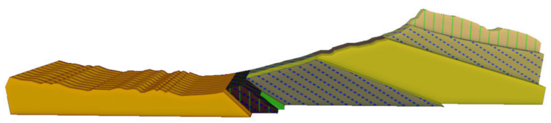
Figure 6: Applications of linear hatch patterns to a generalized virtual 3D city model.

different screen resolutions. The complete scene is visible in the view frustum, and back-face culling is activated. For each test, a total of 100 consecutive frames are rendered and the average rendering performance in frames-per-seconds is tracked. Table 2 shows the results of the performance evaluation.

The performance of the rendering technique depends on the number of hatch layers defined per object. With up to 10 layers for each object, the response time remains within the bounds of real-time interaction. With more layers defined, the response time highly depends on the number of fragments that are processed by the shader.



(a) Three different styles (left to right) for an exploded-view visualization of a virtual 3D dam model that uses grey-scale, linear hatch patterns.



(b) A geologival profile with colored linear hatches visualizing different layers of soil.

Figure 7: Applications of complex linear hatch patterns to virtual 3D architectural and geological models.

Table 1: Geometric complexity of examplary 3D models.

Model	#Obj	#Vertices	#Primitives
City Model	191	14,500	21,366
Dam	10	441	828
Grand Canyon	9	265,726	524,288
Profile	8	3,114	6,194

Table 2: Performance measurements in frames-per-second.

	800×600	1280×960
City Model		
5 layers	52.10	21.85
10 layers	41.69	17.66
40 layers	20.03	9.07
Geological Profile		
5 layers	61.44	31.68
10 layers	60.59	24.64
40 layers	25.59	14.89
Dam		
5 layers	61.06	24.84
10 layers	51.76	19.75
40 layers	22.42	9.11
Grand Canyon		
5 layers	40.22	19.75
10 layers	27.96	12.66
40 layers	8.67	4.12

5 CONCLUSIONS

This paper presents a concept and interactive rendering technique for creating procedural texture pattern for the visualization of 2D and 3D geovirtual environments. It is based on a extensible layer concept that can be easily edited and rendered using consumer graphics hardware. We further analyzed shortcomings and visual artifacts of hatches applied in 3D geovirtual environments and presented three different approaches for counterbalancing these effects. Finally, a variety of application examples are presented, followed by a discussion on performance and limitations of the rendering technique.

ACKNOWLEDGMENTS

This work was funded by the Federal Ministry of Education and Research (BMBF), Germany, within the InnoProfile Transfer research group ”4DnDVis”.

REFERENCES

Akenine-Möller, T., Haines, E., and Hoffman, N. (2008). *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 3rd edition edition.

Amidror, I. (2009). *The Theory of the Moiré Phenomenon: Periodic layers*. Springer.

Bertin, J. (1983). *Semiology of graphics*. University of Wisconsin Press.

Glander, T. and Döllner, J. (2007). Cell-based generalization of 3d building groups with outlier management. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, GIS '07, pages 54:1–54:4, New York, NY, USA. ACM.

Glanville, R. S. (2004). *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter 20 – Texture Bombing. Addison-Wesley Longman.

Goldberg, A., Zwicker, M., and Durand, F. (2008). Anisotropic noise. *ACM Trans. Graph.*, 27(3):54:1–54:8.

Green, C. (2007). Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 9–18, New York, NY, USA. ACM.

Jobst, M. and Döllner, J. (2008). 3d city model visualization with cartography-oriented design. In Schrenk, M., Popovich, V. V., Engelke, D., and Elisei, P., editors, *13th International Conference on Urban Planning, Regional Development and Information Society (REAL CORP)*, pages 507–516. CORP - Competence Center of Urban and Regional Planning.

Kennelly, P. J. and Kimerling, A. J. (2000). Desktop hachure maps from digital elevation models. *Cartographic Perspectives*, (37):78–81.

Kessenich, J., Baldwin, D., and Rost, R. (2010). The OpenGL Shading Language (Version 4.10).

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D., Lewis, J., Perlin, K., and Zwicker, M. (2010). State of the art in procedural noise functions. In Hauser, H. and Reinhard, E., editors, *EG 2010 - State of the Art Reports*. Eurographics, Eurographics Association.

Lewis, J. P. (1989). *Algorithms for solid noise synthesis*, volume 23. ACM Press, New York, New York, USA.

Patel, D., Giertsen, C., Thurmond, J., Gjelberg, J., and Gröller, M. E. (2008). The seismic analyzer: interpreting and illustrating 2D seismic data. *IEEE transactions on visualization and computer graphics*, 14(6):1571–8.

Patel, D., Giertsen, C., Thurmond, J., and Gröller, M. (2007). Illustrative rendering of seismic data. In *Proceeding of vision modeling and visualization*, pages 13–22. Citeseer.

Peachey, D. R. (1985). Solid texturing of complex surfaces. *ACM SIGGRAPH Computer Graphics*, 19(3):279–286.

Perlin, K. (1985). An image synthesizer. *ACM SIGGRAPH Computer Graphics*, 19(3):287–296.

Perlin, K. (2002). Improving noise. *ACM Trans. Graph.*, 21(3):681–682.

Rost, R. J. (2006). *OpenGL® Shading Language*. Addison-Wesley Longman, 2nd ed. edition.

Segal, M., Akeley, K., and Brown, P. (2010). The OpenGL Graphics System: A Specification (Version 4.1).

Wloka, M. (2003). "Batch, Batch, Batch:" What Does It Really Mean? In *Game Developers Conference*.